2

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data needed, and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC 20503.

1. AGENCY USE ONLY *(Leave Blank)*

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED
Final 27 Oct. 1989 to 27 Oct. 1990

4. TITLE AND SUBTITLE Ada Compiler Validation Summary Report: DDC International A/S, DACS for Sun-3/SunOS, Version 4.4 (1.1), SUN-3/60 Workstation (Host) to SUN 3/60 Workstation (Target), 891027S1.10183

5. FUNDING NUMBERS

6. AUTHOR(S)
National Institute of Standards and Technology
Gaithersburg, MD
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Institute of Standards and Technology
National Computer Systems Laboratory
Bldg. 255, Rm. A266
Gaithersburg, MD 20899
USA

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

DDC INTERNATIONAL A/S, DACS for Sun-3/DunOS, Version 4.4 (1.1), Gaithersburg MD, SUN-3/60 Workstation under SunOS UNIX, Version 4.2, Release 4.0_EXPORT (Host & Target), ACVC 1.10.

14. SUBJECT TERMS Ada programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, Validation Testing, Ada Validation Office, Ada Validation Facility, ANSI/MIL-STD-1815A, Ada Joint Program Office

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

90 06 25 116

AVF Control Number: NIST89DDC580_1_1.10
DATE COMPLETED BEFORE ON-SITE: 10-02-89
DATE COMPLETED AFTER ON-SITE: 10-30-89

Ada Compiler Validation Summary Report:

Compiler Name: DACS for Sun-3/SunOS, Version 4.4 (1.1)

Certificate Number: 891027S1.10183

Host: SUN-3/60 Workstation under SunOS UNIX, Version 4.2, Release 4.0_EXPORT

Target: SUN-3/60 Workstation under SunOS UNIX, Version 4.2, Release 4.0_EXPORT

Testing Completed October 27, 1989 Using ACVC 1.10

This report has been reviewed and is approved.

David K. Jefferson

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

L Arnold Johnson

Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

AVF Control Number: NIST89DDC580_1_1.10
DATE VSR COMPLETED BEFORE ON-SITE: 10-02-89
DATE VSR COMPLETED AFTER ON-SITE: 10-30-89
DATE VSR MODIFIED PER AVO COMMENTS: 12-04-89
DATE VSR MODIFIED PER AVO COMMENTS: 04-30-90

Ada COMPILER
VALIDATION SUMMARY REPORT:

Certificate Number: 891027S1.10183
DDC INTERNATIONAL A/S
DACS for Sun-3/SunOS, Version 4.4 (1.1)
SUN-3/60 Workstation Host and SUN-3/60 Workstation Target

Completion of On-Site Testing:
27 October 1989

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT 1-2
1.2 USE OF THIS VALIDATION SUMMARY REPORT 1-2
1.3 REFERENCES 1-3
1.4 DEFINITION OF TERMS 1-3
1.5 ACVC TEST CLASSES 1-4

CHAPTER 2 CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED 2-1
2.2 IMPLEMENTATION CHARACTERISTICS 2-2

CHAPTER 3 TEST INFORMATION

3.1 TEST RESULTS 3-1
3.2 SUMMARY OF TEST RESULTS BY CLASS 3-1
3.3 SUMMARY OF TEST RESULTS BY CHAPTER 3-2
3.4 WITHDRAWN TESTS 3-2
3.5 INAPPLICABLE TESTS 3-2
3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS . . 3-6
3.7 ADDITIONAL TESTING INFORMATION 3-7
3.7.1 Prevalidation 3-7
3.7.2 Test Method 3-7
3.7.3 Test Site 3-8

APPENDIX A CONFORMANCE STATEMENT

APPENDIX B APPENDIX F OF THE Ada STANDARD

APPENDIX C TEST PARAMETERS

APPENDIX D WITHDRAWN TESTS

APPENDIX E COMPILER OPTIONS AS SUPPLIED BY DDC INTERNATIONAL A/S

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of ~~testing~~ this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.
The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

(KR)

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard

- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 27 October 1989 at Lyngby, Denmark.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Software Standards Validation Group
National Computer Systems Laboratory
National Institue of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.

Ada Commentary An Ada Commentary contains all information relevant to the Commentary point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.

Ada Standard ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

Applicant The agency requesting validation.

AVF The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.

AVO The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host The computer on which the compiler resides.

Inapplicable test An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test An ACVC test for which a compiler generates the expected result.

Target The computer which executes the code generated by the compiler.

Test A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn An ACVC test found to be incorrect and not used to check test conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: DACS for Sun-3/SunOS, Version 4.4 (1.1)

ACVC Version: 1.10

Certificate Number: 891027S1.10183

Host Computer:

Machine: SUN-3/60 Workstation

Operating System: SunOS UNIX, Version 4.2, Release 4.0_EXPORT

Memory Size: 8 MBytes

Target Computer:

Machine: SUN-3/60 Workstation

Operating System: SunOS UNIX, Version 4.2, Release 4.0_EXPORT

Memory Size: 8 MBytes

Communications Network: Ethernet (using DNICP net software utility) to the VAX-8350.

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler accepts tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

(1) This implementation supports the additional predefined types SHORT_INTEGER and LONG_FLOAT in the package STANDARD. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) All of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)

(4) NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is gradual. (See tests C45524A..K (11 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round to even. (See tests C46012A..K (11 tests).)

(2) The method used for rounding to longest integer is round to even. (See tests C46012A..K (11 tests).)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

(2) NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

(3) NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when declaring two packed Boolean arrays with INTEGER'LAST + 3 components. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

(6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised before all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

(1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

i. Generics.

(1) Generic specifications and bodies cannot be compiled in separate compilations. (See tests CA2009C, CA2009F, BC3204C, and BC3205D.)

Generic package declarations and bodies can be compiled in separate compilations so long as no instantiations of those units precede the bodies. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected.

(2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

(3) Generic subprogram declarations and bodies can be compiled in separate compilations. (See test CA1012A.)

(4) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)

(5) Generic non-library subprogram bodies cannot be compiled in separate compilations from their stubs. (See test CA2009F.)

(6) Generic package declarations and bodies cannot be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

(7) Generic library package specifications and bodies cannot be compiled in separate compilations. (See tests BC3204C and BC3205D.)

(8) Generic non-library package bodies as subunits cannot be compiled in separate compilations. (See test CA2009C.)

(9) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

j. Input and output.

(1) The package SEQUENTIAL_IO can be instantiated with

unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D and EE2201E.)

(2) The package DIRECT_IO can be instantiated with unconstrained array types but only if the maximum element size supported for DIRECT_IO is 2_147_483_647 bits; otherwise, USE_ERROR is raised. (See tests AE2101H and EE2401D.)

(3) The package DIRECT_IO can be instantiated with record types with discriminants without defaults. (See test EE2401G.)

(4) USE_ERROR is raised when Mode IN_FILE is not supported for the operation of CREATE for SEQUENTIAL_IO. (See test CE2102D.)

(5) USE_ERROR is raised when Mode IN_FILE is not supported for the operation of CREATE for DIRECT_IO. (See test CE2102I.)

(6) USE_ERROR is raised when Mode IN_FILE is not supported for the operation of CREATE for text files. (See test CE3102E.)

(7) Modes IN_FILE and OUT_FILE are supported for text files. (See test CE3102I..K).

(8) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(9) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(10) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)

(11) Overwriting to a sequential file truncates to the last element written. (See test CE2208B.)

(12) Temporary sequential files are given names and deleted when closed. (See test CE2108A.)

(13) Temporary direct files are given names and deleted when closed. (See test CE2108C.)

(14) Temporary text files are given names and deleted when closed. (See test CE3112A.)

(15) More than one internal file can be associated with each external file for sequential files when writing or reading.

(See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

(16) More than one internal file can be associated with each external file for direct files when writing or reading. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)

(17) More than one internal file can be associated with each external file for text files when writing or reading. (See tests CE3111A, CE3111D..E (2 tests), and CE3114B.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 442 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 73 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	123	1131	1888	17	26	46	3231
Inapplicable	6	7	427	0	2	0	442
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	195	572	554	248	172	99	160	331	135	36	250	182	297	3231
Inapplicable	17	77	126	0	0	0	6	1	2	0	2	187	24	442
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

A39005G B97102E C97116A BC3009B CD2A62D CD2A63A
CD2A63B CD2A63C CD2A63D CD2A66A CD2A66B CD2A66C
CD2A66D CD2A73A CD2A73B CD2A73C CD2A73D CD2A76A
CD2A76B CD2A76C CD2A76D CD2A81G CD2A83G CD2A84M
CD2A84N CD2B15C CD2D11B CD5007B CD5011O CD7105A
CD7203B CD7204B CD7205C CD7205D CE2107I CE3111C
CE3301A CE3411B E28005C ED7004B ED7005C ED7005D
ED7006C ED7006D

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 442 tests were inapplicable for the reasons indicated:

a. The following 201 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..Y (14 tests) C35705L..Y (14 tests)
C35706L..Y (14 tests) C35707L..Y (14 tests)

C35708L..Y (14 tests)
C45241L..Y (14 tests)
C45421L..Y (14 tests)
C45524L..Z (15 tests)
C45641L..Y (14 tests)
C35802L..Z (15 tests)
C45321L..Y (14 tests)
C45521L..Z (15 tests)
C45621L..Z (15 tests)
C46012L..Z (15 tests)

b. C24113I..K (3 tests) are not applicable because the line length of the input file must not exceed 126 characters.

c. C35508I, C35508J, C35508M, C35508N, AD1C04D, AD3015C, AD3015F, AD3015H, AD3015K, CD1C04B, CD1C04C, CD1C04E, CD2A23C, CD2A23D, CD2A24C, CD2A24D, CD2A24G, CD2A24H, CD3015A, CD3015B, CD3015D, CD3015E, CD3015G, CD3015I, CD3015J, CD3015L, CD4051A, CD4051B, CD4051C, CD4051D (30 tests) are not applicable because this implementation does not support the specified change in representation for derived types.

d. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

e. A39005E, C87B62C, CD1009L, CD1C03F, CD2D11A, CD2D13A, ED2A56A (7 tests) are not applicable because 'SMALL clause is not supported.

f. The following 16 tests are not applicable because this implementation does not support a predefined type LONG_INTEGER:

C45231C C45304C C45502C C45503C C45504C
C45504F C45611C C45613C C45614C C45631C
C45632C B52004D C55B07A B55B09C B86001W
CD7101F

g. C45231D, CD7101G, and B86001X, are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, or SHORT_INTEGER.

h. C45531M, C45531N, C45532M, and C45532N use fine 48 bit fixed point base types which are not supported by this compiler.

i. C45531O, C45531P, C45532O, and C45532P use coarse 48 bit fixed point base types which are not supported by this compiler.

j. C4A013B is not applicable because the evaluation of an expression involving 'MACHINE_RADIX applied to the most precise floating-point type would raise an exception; since the expression must be static, it is rejected at compile time.

k. B86001X and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER or SHORT_INTEGER.

l. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

n. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, or LONG_FLOAT.

p. C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

q. CA2009C is not applicable because this implementation does not permit compilation of generic non-library package bodies in separate files from their specifications.

r. CA2009F is not applicable because this implementation does not permit compilation of generic non-library subprogram bodies in separate files from their specifications.

s. BC3204C and BC3205D are not applicable because this implementation does not permit compilation of generic library package bodies in separate files from their specifications.

t. CD1009C, CD2A41A..B, CD2A41E, CD2A42A..J (14 tests) are not applicable because this implementation does not support the 'SIZE clause for floating-point types.

u. CD2A51C, CD2A52A..D, CD2A52G..J, CD2A53A..E, CD2A54A..D, CD2A54G..J (22 tests) are not applicable because this implementation does not support the 'SIZE clause for a fixed-point types.

v. CD2A61A..F, CD2A61H..L, CD2A62A..C, CD2A64A..C, CD2A65A..C, CD2A71A..D, CD2A72A..D, CD2A74A..B, CD2A75A..B (32 tests) are not applicable because this implementation does not support the 'SIZE clause for an array type which does not imply compression of inter-component gaps.

w. CD2A84B..I and CD2A84K..L (10 tests) are not applicable because this implementation does not support the 'SIZE clause for an access type.

x. CD4041A is not applicable because this implementation does not support the alignment clauses for alignments other than SYSTEM.STORAGE_UNIT for record representation clauses.

y. CD5003B..I, CD5011A, CD5011C, CD5011E, CD5011G, CD5011I, CD5011K, CD5011M, CD5011Q, CD5012A..B, CD5012E..F, CD5012I, CD5012M, CD5013A, CD5013C, CD5013E, CD5013G, CD5013I, CD5013K, CD5013M, CD5013O, CD5014A, CD5014C, CD5014E, CD5014G, CD5014I, CD5014K, CD5014M, CD5014O, CD5014R, CD5014T, CD5014V..Z (44 tests) are not applicable because this implementation does not support address clauses for a variable.

z. CD5011B, CD5011D, CD5011F, CD5011H, CD5011L, CD5011N, CD5011R, CD5011S, CD5012C..D, CD5012G..H, CD5012L, CD5013B, CD5013D, CD5013F, CD5013H, CD5013L, CD5013N, CD5013R, CD5014B, CD5014D, CD5014F, CD5014H, CD5014J, CD5014L, CD5014N, CD5014U (28 tests) are not applicable because this implementation does not support address clauses for a constant.

aa. CD5012J, CD5013S, CD5014S (3 tests) are not applicable because this implementation does not support address clauses.

ab. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

ac. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

ad. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

ae. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

af. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

ag. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

ah. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

ai. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

aj. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

ak. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

al. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

am. CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.

an. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

ao. CE2105A is inapplicable because CREATE with IN_FILE mode is not supported by this implementation for SEQUENTIAL_IO.

ap. CE2105B is inapplicable because CREATE with IN_FILE mode is not supported by this implementation for DIRECT_IO.

aq. CE3102F is inapplicable because text file RESET is supported by this implementation.

ar. CE3102G is inapplicable because text file deletion of an external file is supported by this implementation.

as. CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.

at. CE3102J is inapplicable because text file OPEN with IN_FILE mode is supported by this implementation.

au. CE3102K is inapplicable because text file OPEN with OUT_FILE mode is supported by this implementation.

av. CE3109A is inapplicable because text file CREATE with IN_FILE mode is not supported by this implementation.

aw. CE3111B and CE3115A simultaneously associate input and output files with the same external file, and expect that output is immediately written to the external file and available for reading; this implementation buffers files, and each test's attempt to read such output (at lines 87 & 101, respectively) raises END_ERROR.

ax. EE2401D is inapplicable because the maximum element size supported for DIRECT_IO is 2_147_483_647 bits. USE_ERROR is raised.

3.66 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 73 tests.

The following 65 tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A B26001A B26002A B26005A B28001D B28003A B29001A
B2A003A B2A003B B2A003C B33301A B35101A B37106A B37301B
B37302A B38003A B38003B B38009A B38009B B51001A B53009A
B54A01C B54A01H B55A01A B61001C B61001D B61001F B61001H

B61001I B61001M B61001R B61001S B61001W B67001H B91001A
B91002A B91002B B91002C B91002D B91002E B91002F B91002G
B91002H B91002I B91002J B91002K B91002L B95030A B95061A
B95061F B95061G B95077A B97103E B97104G BA1101B BC1109A
BC1109C BC1109D BC1202A BC1202B BC1202E BC1202F BC1202G
BC2001D BC2001E

The following 8 tests contain modifications to their respective source code files:

C34007A, C34007D, C34007G, C34007J, C34007M, C34007P, C34007S, and C87B62B (8 tests) The AVO accepts the implementer's argument that, without there being a STORAGE_SIZE length clause for an access type, the meaning of the attribute 'STORAGE_SIZE is undefined for that type. Therefore, a length clause has been added in these tests in order to alter the default size of a collection. 1024 was used for all of the above tests except for C34007D and C34007G which used 2048.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the DACS for Sun-3/SunOS, Version 4.4 (1.1) compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the DACS for Sun-3/SunOS, Version 4.4 (1.1) compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	SUN-3/60 Workstation
Host operating system:	SunOS UNIX, Version 4.2, Release 4.0_EXPORT
Target computer:	SUN-3/60 Workstation
Target operating system:	SunOS UNIX, Version 4.2, Release 4.0_EXPORT
Compiler:	DACS for Sun-3/SunOS, Version 4.4 (1.1)

The ACVC Test Suite was loaded onto a VAX-8350 from the magnetic tape. The ACVC Test Suite was then downloaded onto the SUN-3/60 Workstation from the VAX-8530 via Ethernet (using DNICP net software utility).

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized on-site. Tests requiring modifications during the prevalidation testing were modified on-site.

TEST INFORMATION

The contents of the magnetic tape were loaded onto a VAX-8350 and transferred to the host computer, SUN-3/60 Workstation, via Ethernet (using DNICP net software utility).

After the test files were loaded to disk, the full set of tests was compiled and linked on the SUN-3/60 Workstation, and all executable tests were run on the SUN-3/60 Workstation. Results were transferred from the SUN-3/60 Workstation to the VAX-8530 via Ethernet (using DNICP net software utility). The results were then printed from the VAX-8350 computer.

The compiler was tested using command scripts provided by DDC INTERNATIONAL A/S and reviewed by the validation team. The compiler was tested using the following option settings. See Appendix E for a complete listing of the compiler options for this implementation.

```
-L
-a
```

Tests were compiled, linked, and executed (as appropriate) using a single host and target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. Selected listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Lyngby, Denmark and was completed on 27 October 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

DDC INTERNATIONAL A/S has submitted the following Declaration of Conformance concerning the DACS for Sun-3/SunOS, Version 4.4 (1.1).

DECLARATION OF CONFORMANCE

Compiler Implementor:

DDC International A/S
Gl. Lundtoftevej 1B
2800 Lyngby, Denmark

Ada Validation Facility:

Ada Validation Facility
National Computer Systems Laboratory (NCSL)
National Institute of Standards and Technology
Building 225, Room A266
Gaitherburg, MD 20899, U.S.A.

Ada Compiler Validation Capability (ACVC) Version: 1.10

	Base Configuration
Base Compiler Name:	DACS for Sun-3/SunOS, Version 4.4 (1.1)
Host Architecture:	Sun-3/60 Workstation
Host OS and Version:	SunOS UNIX, Version 4.2, Release 4.0_Export
Target Architecture:	Same as host
Target OS and Version:	Same as host

Implementor's Declaration

I, the undersigned, representing DDC International A/S, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that DDC International A/S is the owner of record of the Ada language compiler(s) listed above, and as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

Date: 25 October 1989

DDC International A/S
Hasse Hansson, Department Manager

Owner's Declaration

I, the undersigned, representing DDC International A/S, take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/taget performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Date: 25 October 1989

DDC International A/S
Hasse Hansson, Department Manager

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the DACS for Sun-3/SunOS, Version 4.4 (1.1) compiler, as described in this Appendix, are provided by DDC INTERNATIONAL A/S. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

...

 type SHORT_INTEGER is range -32_768 .. 32_767;
 type INTEGER is range -2_147_483_648 .. 2_147_483_647;


 type FLOAT is digits 6 range
      -16#7.FFFF_C#E31 .. 16#7.FFFF_C#E31;
 type LONG_FLOAT is digits 15 range
      -16#F.FFFF_FFFF_FFFF#E255 .. 16#F.FFFF_FFFF_FFFF#E255;

 type DURATION is delta 2**(-14) range
      -131_072.00000 .. 131_071.00000 ;

...

end STANDARD;
```

F Appendix F of the Ada Reference Manual

F.0 Introduction

This appendix describes the implementation-dependent characteristics of the DDC-I Sun-3/SunOS V Ada Compiler, as required in the Appendix F frame of the Ada Reference Manual (ANSI/MIL-STD 1815A).

F.1 Implementation-Dependent Pragmas

There is one implementation-defined pragma: Interface_spelling, see section 5.6.6.2.

F.2 Implementation-Dependent Attributes

No implementation-dependent attributes are defined.

F.3 Package SYSTEM

```
pragma page;
package SYSTEM is

   type ADDRESS             is new  INTEGER;
   subtype PRIORITY         is INTEGER range 1 .. 32;
   type NAME                is ( SUN );
   SYSTEM_NAME:             constant NAME  := SUN;
   STORAGE_UNIT:            constant       := 8;
   MEMORY_SIZE:             constant       := 2048 * 1024;
   MIN_INT:                 constant       := -2_147_483_648;
   MAX_INT:                 constant       := 2_147_483_647;
   MAX_DIGITS:              constant       := 15;
   MAX_MANTISSA:            constant       := 31;

   FINE_DELTA:              constant       := 2#1.0#E-31;
   TICK:                    constant       := 1.0;

   type interface_language is (C,AS);

-- Compiler system dependent types:

    SUBTYPE Integer_16   IS short_integer;
    SUBTYPE Natural_16   IS Integer_16 RANGE 0..Integer_16'LAST;
    SUBTYPE Positive_16  IS Integer_16 RANGE 1..Integer_16'LAST;

    SUBTYPE Integer_32   IS integer;
    SUBTYPE Natural_32   IS Integer_32 RANGE 0..Integer_32'LAST;
    SUBTYPE Positive_32  IS Integer_32 RANGE 1..Integer_32'LAST;


end SYSTEM;
```

F.4 Representation Clauses

F.4.1 Length Clause

A size attribute for a type T is accepted in the following cases:

- If T is a discrete type then the specified size must be greater than an equal to the number of bits needed to represent a value of the type, and less than or equal to 32.
- If T is a fixed point type, a floating point type, an access type or a task type the specified size must be equal to the number of bits used to represent values of the type.
- If T is a record type that is not derived then the specified size must be greater than or equal to the number of bits used to represent value of the type.
- If T is an array type that is not derived, and has a size known at compile time then the specified size must be equal to the number of bits used to represent values of the type. In all other cases the size attribute is not accepted.

Furthermore, the size attribute has only effect if the type is part of a composite type.

- Using the STORAGE_SIZE attribute for a collection will set an upper limit on the total size of objects allocated in this collection. If further allocation is attempted, the exception STORAGE_ERROR is raised.
- When STORAGE_SIZE is specified in a length clause for a task, the process stack area will be of the specified size. The process stack area will be allocated inside the "standard" stack segment.

F.4.2 Enumeration Representation Clause

Enumeration representation clauses may specify representations in the range of INTEGER'FIRST + 1..INTEGER'LAST - 1.

Enumeration representation clauses are not supported for derived types.

F.4.3 Record Representation Clauses

When representation clauses are applied to records the following restrictions are imposed:

- the component type is a discrete type different from LONG_INTEGER,

- the component type is an array with a discrete element type different from LONG_INTEGER,

- if the component is a record or an unpacked array, it must start on a storage unit boundary, a storage unit being 16 bits,

- a record occupies an integral number of storage units,

- a record must be specified with its proper size (in bits), regardless of whether the component is an array or not,

- if a non-array component has a size which equals or exceeds one storage unit (16 bits), the component must start on a storage unit boundary, i.e. the component must be specified as:

  ```
  component        at N range 0..16 * M - 1;
  ```

  where N specifies the relative storage unit number (0,1,...) from the beginning of the record, and M the required number of storage units (1,2,...)

- the elements in an array component should always be wholly contained in one storage unit,

- if a component has a size which is less than one storage unit, it must be wholly contained within a single storage unit:

  ```
  component        at N range X .. Y;
  ```

  where N is as in the previous paragraph, and 0 <= X <= Y <= 15

If the record type contains components which are not covered by a component clause, they are allocated consecutively after the component with the value. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler.

F.4.3.1 Alignment Clauses

Alignment clauses for records are implemented with the following characteristics:

- If the declaration of the record type is done at the outermost level in a library package, any alignment is accepted.
- If the record declaration is done at a given static level (higher than the outermost library level, i.e. the permanent area), only word alignments are accepted.
- Any record object declared at the outermost level in a library package will be aligned according to the alignment clause specified for the type. Record objects declared elsewhere can only be aligned on a word boundary. If the record type has been associated a different alignment, an error message will be issued.
- If a record type with an associated alignment clause is used in a composite type, the alignment is required to be one word: an error message is issued if this is not the case.

F.5 Implementation-Dependent Names for Implementation-Dependent Components

None defined by the compiler.

F.6 Address Clauses

Not supported by the compiler.

F.7 Unchecked Conversion

Unchecked conversion is only allowed between objects of the same "size". In this context the "size" of an array is equal to that of two access values and the "size" of a packed array is equal to two access values and an integer. This is the only restriction imposed on unchecked conversion.

F.8 Input-Output Packages

The implementation supports all requirements of the Ada language. It is an effective interface to the UNIX file system, and in the case of text input-output also an effective interface to the UNIX standard input, standard output and standard error streams.

This section describes the functional aspects of the interface to the UNIX file system, including the means by which the various file control facilities are made available to the Ada programmer.

The Ada input-output concept as defined in Chapter 14 of the ARM does not constitute a complete functional specification of the input-output packages. Some aspects are not discussed at all, while others are deliberately left open to an implementation. These gaps are filled by this section.

The reader should be familiar with

[DoD 83] - The Ada Language definition

and some sections require that the reader is familiar with

[UNIX 3] - UNIX Programmer Reference Manual

F.8.1 External Files

External files can be on disc, tape, or be a character device (a line printer, terminal etc.).

Files on disc exist after the execution of the program unless given an empty NAME parameter.

The implementation will raise USE_ERROR when an operation is inappropriate for the physical device. In particular the concept of a page or end-of-file or file size are not considered to be applicable to terminal devices and attempted use of operations involving these concepts will raise USE_ERROR.

Deletion is not allowed on non-disc devices and requires write access.

Creation of files with mode IN_FILE will raise USE_ERORR.

F.8.2 File Management

This subsection contains information regarding file management:

- restriction on sequential and direct input-output,
- the NAME parameter,
- the FORM parameter,
- file access.

F.8.2.1 Restrictions on Sequential and Direct Input-Output

The only restriction is that placed on the element size, i.e. the number of bytes occupied by the ELEMENT_TYPE: the maximum size allowed is 2_147_1183_647 bits; and if the size of the type is variable, the maximum size must be determinable at the point of instantiation from the value of the SIZE attribute for the element type.

F.8.2.2 The NAME Parameter

The NAME parameter when non-empty must be valid UNIX path name. Access denial to any directory in the path name will raise USE_ERROR.

The UNIX names "stdin", "stdout", and "stderr" can be used in conjunction with TEXT_IO.open. No physical opening of the external file is performed and the Ada file will be associated with the already open external file.

Temporary files (NAME = "") are created using tmpname (3) and will be deleted on closure. Abnormal program termination may leave temporary files in existence.

Default naming conventions and version numbers are not applicable to UNIX.

F.8.2.3 The FORM Parameter

The FORM parameter has the following facilities:

a) Opening a FIFO special file using open(2) system call. This is achieved by the string "FIFO". If this facility is used with CREATE, the exception USE_ERROR will be raised. This facility is not available for direct_io or text_io and raises USE_ERROR.

The default for this facility is indicated by the "ORDINARY" string designating the creation of an ordinary file. If this string is used with OPEN and the external file is of type FIFO special, the operation raises USE_ERROR.

The O_NDELAY flag associated with FIFO specials (see open(2)) can be modified using an additional string after the "FIFO" string. The strings "O_NDELAY=ON" and "O_NDELAY=OFF" set the flag on and off respectively. The default is "O_NDELAY=OFF". Thus "FIFO O_NDELAY=ON" opens a FIFO special file and set the O_NDELAY flag on.

b) The use of the string "APPEND" with text-files prevents the emptying of the file for the OPEN operation. The presence of "APPEND" in the form parameter is only applicable to OPEN, and its use in CREATE will raise USE_ERROR. The string "NOAPPEND" signifies the default.

The opened file will be treated by the routines delete as if empty.

Opening direct and sequential files with the "APPEND" or "NOAPPEND" raises USE_ERROR.

c) The changing of default access rights by specifying the mode parameter used in the open(2) system call used to implement the Ada CREATE procedure. This is achieved by use of the string "MODE=<mode>" where <mode> is an octal, decimal or hexadecimal integer in the standard UNIX format. Only the nine least significant bits of the creation mask are used. This facility is also used by OPEN to change the access permissions by means of the chmod(2) system call.

The default for mode is 0644, allowing the owner to read and write, and the group and others to read. The bits mean (as in standard UNIX):

```
rw- r-- r--
 │   │   │
 │   │   └ Other privileges
 │   │
 │   └ Group privileges
 │
 └ Owner privileges
```

NOTES:

The options are delimited by commas.

If more than one of the three option types is included, the rightmost option is selected.

Blanks are not significant in any part of the string.

The FORM parameter provides all default options as required in the ARM.

F.8.2.3.1 Syntax of the Form Parameter

```
<form_parameter> := [<option> [,<option> [,<option>] ] ]

<option> := <access_rights> | <fifo_option> | <append_option>

<access_rights> := MODE= <mode>

<fifo_option> := <fifo_special> | ORDINARY

<append_option> := APPEND | NOAPPEND

<mode> := <hex_number> | <octal_number> | <decimal_number>

<fifo_special> := FIFO [<o_ndelay_parameter>]

<o_ndelay_parameter> := O_NDELAY=ON | O_NDELAY=OFF


<decimal_number> := <decimal_digit> {<decimal_digit>}

<hex_number>     := 0 x <hex_suffix>

<octal_number>   := 0 <octal_suffix>

<hex_suffix>     := <hex_digit> {<hex_digit>}

<octal_suffix>   := <octal_digit> {<octal_digit>}

<hex_digit>      := 0 | 1 ...| 9 | A | ... | F | a | ...| f

<decimal_digit>  := 0 | 1 ...| 9

<octal_digit>    := 0 | 1 ...| 7
```

F.8.2.4 File Access

Any number of files in an Ada program may be associated with any external file at any time. Each end of a FIFO special file must be accessed from two UNIX processes which will have to correspond to two Ada programs.

It is the responsibility of the programmer to consider the effects of file sharing between programs.

The RESET and OPEN operations to OUT_FILE mode empty the file in SEQUENTIAL_IO and TEXT_IO.

Interchanging between SEQUENTIAL_IO and DIRECT_IO for files of the same object types can be achieved without taking special measures.

The state of the external file at any moment is in general undefined. Closing and resetting a file will, however, flush any buffering in the input-output packages. Unpredictable results may occur if the program is terminated without calling CLOSE.

F.8.3 Sequential Input-Output

The implementation omits type checking for DATA_ERROR, in case the element type is of an unconstrained type, ARM 14.2.2(4), i.e.:

```
... f : FILE_TYPE
type et  is 1..100;
type eat is array( et range <> ) of integer;

X : eat( 1..2 );
Y : eat( 1..4 );
...
-- write X, Y:

write( f, X); write( f, Y); reset( f, IN_FILE);

-- read X into Y and Y into X:

read( f, Y); read( f, X);
```

This will give undefined values in the last 2 elements of Y, and not DATA_ERROR.

F.8.3.1 Specification of the Package Sequential_IO

```
with BASIC_IO_TYPES;

with IO_EXCEPTIONS;

generic

   type ELEMENT_TYPE is private;

package SEQUENTIAL_IO is

  type FILE_TYPE is limited private;

  type FILE_MODE is (IN_FILE, OUT_FILE);

-- File management

   procedure CREATE(FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE := OUT_FILE;
                    NAME : in     STRING    := "";
                    FORM : in     STRING    := "");

   procedure OPEN  (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE;
                    NAME : in     STRING;
                    FORM : in     STRING := "");

   procedure CLOSE (FILE : in out FILE_TYPE);

   procedure DELETE(FILE : in out FILE_TYPE);

   procedure RESET (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE);

   procedure RESET (FILE : in out FILE_TYPE);

   function MODE   (FILE : in FILE_TYPE) return FILE_MODE;

   function NAME   (FILE : in FILE_TYPE) return STRING;

   function FORM   (FILE : in FILE_TYPE) return STRING;

   function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

-- input and output operations

   procedure READ  (FILE : in     FILE_TYPE;
                    ITEM :    out ELEMENT_TYPE);

   procedure WRITE (FILE : in FILE_TYPE;
                    ITEM : in ELEMENT_TYPE);
   function END_OF_FILE(FILE : in FILE_TYPE) return BOOLEAN;

-- exceptions

   STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;

private

   type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end SEQUENTIAL_IO;
```

F.8.4 Direct Input-Output

The implementation omits type checking for DATA_ERROR, in case the element type is of an unconstrained type, [Dod 83] 14.2.4(4), see F.8.3.

F.8.4.1 Specification of the Package Direct_IO

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;

generic

   type ELEMENT_TYPE is private;

package DIRECT_IO is


   type FILE_TYPE is limited private;

   type FILE_MODE is (IN_FILE, INOUT_FILE, OUT_FILE);

   type COUNT is range 0..INTEGER'LAST;
   subtype POSITIVE_COUNT is COUNT range 1..COUNT'LAST;


-- File management

   procedure CREATE(FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE  := INOUT_FILE;
                    NAME : in     STRING     := "";
                    FORM : in     STRING     := "");

   procedure OPEN  (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE;
                    NAME : in     STRING;
                    FORM : in     STRING    := "");

   procedure CLOSE (FILE : in out FILE_TYPE);

   procedure DELETE(FILE : in out FILE_TYPE);

   procedure RESET (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE);

   procedure RESET (FILE : in out FILE_TYPE);

   function MODE   (FILE : in FILE_TYPE) return FILE_MODE;

   function NAME   (FILE : in FILE_TYPE) return STRING;

   function FORM   (FILE : in FILE_TYPE) return STRING;

   function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

-- input and output operations
```

```
   procedure READ  (FILE : in       FILE_TYPE;
                    ITEM :      out ELEMENT_TYPE;
                    FROM : in       POSITIVE_COUNT);
   procedure READ  (FILE : in       FILE_TYPE;
                    ITEM :      out ELEMENT_TYPE);

   procedure WRITE (FILE : in FILE_TYPE;
                    ITEM : in ELEMENT_TYPE;
                    TO   : in POSITIVE_COUNT);
   procedure WRITE (FILE : in FILE_TYPE;
                    ITEM : in ELEMENT_TYPE);

   procedure SET_INDEX(FILE : in FILE_TYPE;
                       TO   : in POSITIVE_COUNT);

   function INDEX(FILE : in FILE_TYPE) return POSITIVE_COUNT;

   function SIZE (FILE : in FILE_TYPE) return COUNT;

   function END_OF_FILE(FILE : in FILE_TYPE) return BOOLEAN;


-- exceptions

   STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;

private

  type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end DIRECT_IO;
```

F.8.5 Specification of the Package Text_IO

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;
package TEXT_IO is

   type FILE_TYPE is limited private;

   type FILE_MODE is (IN_FILE, OUT_FILE);

   type    COUNT is range 0 .. INTEGER'LAST;
   subtype POSITIVE_COUNT is COUNT range 1 .. COUNT'LAST;
   UNBOUNDED: constant COUNT:= 0; -- line and page length

   subtype FIELD          is INTEGER range 0 .. 35;

   subtype NUMBER_BASE    is INTEGER range 2 .. 16;

   type TYPE_SET is (LOWER_CASE, UPPER_CASE);

   -- File Management

   procedure CREATE (FILE  : in out FILE_TYPE;
                     MODE  : in     FILE_MODE := OUT_FILE;
                     NAME  : in     STRING    := "";
                     FORM  : in     STRING    := "");

   procedure OPEN   (FILE  : in out FILE_TYPE;
                     MODE  : in     FILE_MODE;
                     NAME  : in     STRING;
                     FORM  : in     STRING    := "");

   procedure CLOSE  (FILE  : in out FILE_TYPE);
   procedure DELETE (FILE  : in out FILE_TYPE);
   procedure RESET  (FILE  : in out FILE_TYPE;
                     MODE  : in     FILE_MODE);
   procedure RESET  (FILE  : in out FILE_TYPE);

   function  MODE   (FILE  : in FILE_TYPE) return FILE_MODE;
   function  NAME   (FILE  : in FILE_TYPE) return STRING;
   function  FORM   (FILE  : in FILE_TYPE) return STRING;

   function  IS_OPEN(FILE  : in FILE_TYPE) return BOOLEAN;

   -- Control of default input and output files

   procedure  SET_INPUT   (FILE : in FILE_TYPE);
   procedure  SET_OUTPUT  (FILE : in FILE_TYPE);

   function   STANDARD_INPUT   return FILE_TYPE;
   function   STANDARD_OUTPUT  return FILE_TYPE;
function   CURRENT_INPUT    return FILE_TYPE;
function   CURRENT_OUTPUT   return FILE_TYPE;


-- specification of line and page lengths

procedure SET_LINE_LENGTH  (FILE : in FILE_TYPE;
                            TO   : in COUNT);
procedure SET_LINE_LENGTH  (TO   : in COUNT);

procedure SET_PAGE_LENGTH  (FILE : in FILE_TYPE;
                            TO   : in COUNT);
procedure SET_PAGE_LENGTH  (TO   : in COUNT);

function  LINE_LENGTH      (FILE : in FILE_TYPE) return
                                                 COUNT;
function  LINE_LENGTH                            return
                                                 COUNT;

function  PAGE_LENGTH      (FILE : in FILE_TYPE) return
                                                 COUNT;
function  PAGE_LENGTH                            return
                                                 COUNT;

-- Column, Line, and Page Control

procedure NEW_LINE    (FILE    : in FILE_TYPE;
                       SPACING : in POSITIVE_COUNT := 1);
procedure NEW_LINE    (SPACING : in POSITIVE_COUNT := 1);

procedure SKIP_LINE   (FILE    : in FILE_TYPE;
                       SPACING : in POSITIVE_COUNT := 1);
procedure SKIP_LINE   (SPACING : in POSITIVE_COUNT := 1);

function  END_OF_LINE (FILE : in FILE_TYPE) return
                                            BOOLEAN;
function  END_OF_LINE                       return
                                            BOOLEAN;

procedure NEW_PAGE    (FILE : in FILE_TYPE);
procedure NEW_PAGE                         ;

procedure SKIP_PAGE   (FILE : in FILE_TYPE);
procedure SKIP_PAGE                        ;

function  END_OF_PAGE (FILE : in FILE_TYPE) return
                                            BOOLEAN;
function  END_OF_PAGE                       return
                                            BOOLEAN;

function  END_OF_FILE (FILE : in FILE_TYPE) return
                                            BOOLEAN;
```

```
function  END_OF_FILE                          return
                                               BOOLEAN;

procedure SET_COL      (FILE : in FILE_TYPE;
                        TO   : in POSITIVE_COUNT);
procedure SET_COL      (TO   : in POSITIVE_COUNT);

procedure SET_LINE     (FILE : in FILE_TYPE;
                        TO   : in POSITIVE_COUNT);
procedure SET_LINE     (TO   : in POSITIVE_COUNT);

function  COL          (FILE : in FILE_TYPE) return
                                  POSITIVE_COUNT;
function  COL                                return
                                  POSITIVE_COUNT;

function  LINE         (FILE : in FILE_TYPE) return
                                  POSITIVE_COUNT;
function  LINE                               return
                                  POSITIVE_COUNT;

function  PAGE         (FILE : in FILE_TYPE) return
                                  POSITIVE_COUNT;
function  PAGE                               return
                                  POSITIVE_COUNT;

-- Character Input-Output

procedure GET  (FILE : in      FILE_TYPE;
                ITEM :    out CHARACTER);
procedure GET  (ITEM :    out CHARACTER);
procedure PUT  (FILE : in FILE_TYPE;
                ITEM : in CHARACTER);
procedure PUT  (ITEM : in CHARACTER);

-- String Input-Output

procedure GET  (FILE : in      FILE_TYPE;
                ITEM :    out STRING);
procedure GET  (ITEM :    out STRING);
procedure PUT  (FILE : in FILE_TYPE;
                ITEM : in STRING);
procedure PUT  (ITEM : in STRING);

procedure GET_LINE  (FILE : in      FILE_TYPE;
                     ITEM :    out STRING;
                     LAST :    out NATURAL);
procedure GET_LINE  (ITEM :    out STRING;
                     LAST :    out NATURAL);
procedure PUT_LINE  (FILE : in      FILE_TYPE;
                     ITEM : in      STRING);
procedure PUT_LINE  (ITEM : in      STRING);
```

```
 -- Generic Package for Input-Output of Integer Types

generic
    type NUM is range <>;
package INTEGER_IO is

   DEFAULT_WIDTH : FIELD        := NUM'WIDTH;
   DEFAULT_BASE  : NUMBER_BASE  :=       10;

   procedure GET  (FILE  : in      FILE_TYPE;
                   ITEM  :    out NUM;
                   WIDTH : in      FIELD := 0);
   procedure GET  (ITEM  :    out NUM;
                   WIDTH : in      FIELD := 0);

   procedure PUT  (FILE  : in FILE_TYPE;
                   ITEM  : in NUM;
                   WIDTH : in FIELD := DEFAULT_WIDTH;
                   BASE  : in NUMBER_BASE := DEFAULT_BASE);
   procedure PUT  (ITEM  : in NUM;
                   WIDTH : in FIELD := DEFAULT_WIDTH;
                   BASE  : in NUMBER_BASE := DEFAULT_BASE);

   procedure GET  (FROM  : in      STRING;
                   ITEM  :    out NUM;
                   LAST  :    out POSITIVE);
   procedure PUT  (TO    :    out STRING;
                   ITEM  : in      NUM;
                   BASE  : in      NUMBER_BASE :=
                                              DEFAULT_BASE);

end INTEGER_IO;
```

```
   -- Generic Packages for Input-Output of Real Types

generic
   type NUM is digits <>;
package FLOAT_IO is

   DEFAULT_FORE : FIELD :=                2;
   DEFAULT_AFT  : FIELD := NUM'digits - 1;
   DEFAULT_EXP  : FIELD :=                3;

   procedure GET  (FILE  : in      FILE_TYPE;
                   ITEM  :     out NUM;
                   WIDTH : in      FIELD := 0);
   procedure GET  (ITEM  :     out NUM;
                   WIDTH : in      FIELD := 0);

   procedure PUT  (FILE : in FILE_TYPE;
                   ITEM : in NUM;
                   FORE : in FIELD := DEFAULT_FORE;
                   AFT  : in FIELD := DEFAULT_AFT;
                   EXP  : in FIELD := DEFAULT_EXP);
   procedure PUT  (ITEM : in NUM;
                   FORE : in FIELD := DEFAULT_FORE;
                   AFT  : in FIELD := DEFAULT_AFT;
                   EXP  : in FIELD := DEFAULT_EXP);

   procedure GET  (FROM : in      STRING;
                   ITEM :     out NUM;
                   LAST :     out POSITIVE);
   procedure PUT  (TO   :     out STRING;
                   ITEM : in      NUM;
                   AFT  : in      FIELD := DEFAULT_AFT;
                   EXP  : in      FIELD := DEFAULT_EXP);

end FLOAT_IO;
```

```
generic
   type NUM is delta <>;
package FIXED_IO is

   DEFAULT_FORE : FIELD := NUM'FORE;
   DEFAULT_AFT  : FIELD := NUM'AFT;
   DEFAULT_EXP  : FIELD := 0;

   procedure GET  (FILE  : in      FILE_TYPE;
                   ITEM  :     out NUM;
                   WIDTH : in      FIELD := 0);
   procedure GET  (ITEM  :     out NUM;
                   WIDTH : in      FIELD := 0);

   procedure PUT  (FILE : in FILE_TYPE;
                   ITEM : in NUM;
                   FORE : in FIELD := DEFAULT_FORE;
                   AFT  : in FIELD := DEFAULT_AFT;
                   EXP  : in FIELD := DEFAULT_EXP);

   procedure PUT  (ITEM : in NUM;
                   FORE : in FIELD := DEFAULT_FORE;
                   AFT  : in FIELD := DEFAULT_AFT;
                   EXP  : in FIELD := DEFAULT_EXP);

   procedure GET  (FROM : in     STRING;
                   ITEM :    out NUM;
                   LAST :    out POSITIVE);
   procedure PUT  (TO   :    out STRING;
                   ITEM : in     NUM;
                   AFT  : in     FIELD := DEFAULT_AFT;
                   EXP  : in     FIELD := DEFAULT_EXP);

end FIXED_IO;

   -- Generic Package for Input-Output of Enumeration Types
```

```
generic
   type ENUM is (<>);
package ENUMERATION_IO is

   DEFAULT_WIDTH   : FIELD    := 0;
   DEFAULT_SETTING : TYPE_SET := UPPER_CASE;

   procedure GET  (FILE : in      FILE_TYPE;
                   ITEM :     out ENUM);
   procedure GET  (ITEM :     out ENUM);

   procedure PUT  (FILE  : in FILE_TYPE;
                   ITEM  : in ENUM;
                   WIDTH : in FIELD       := DEFAULT_WIDTH;
                   SET   : in TYPE_SET    := DEFAULT_SETTING);

   procedure PUT  (ITEM  : in ENUM;
                   WIDTH : in FIELD       := DEFAULT_WIDTH;
                   SET   : in TYPE_SET    := DEFAULT_SETTING);

   procedure GET  (FROM : in      STRING;
                   ITEM :     out ENUM;
                   LAST :     out POSITIVE);
   procedure PUT  (TO   :     out STRING;
                   ITEM : in      ENUM;
                   SET  : in      TYPE_SET := DEFAULT_SETTING);

end ENUMERATION_IO;

   -- Exceptions

   STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;
   LAYOUT_ERROR : exception renames IO_EXCEPTIONS.LAYOUT_ERROR;

private

  type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end TEXT_IO;
```

F.8.6 Low Level Input-Output

The package LOW_LEVEL_IO is empty.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	1..125 => 'A', 126 => '1'
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	1..125 => 'A', 126 => '2'
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	1..63 => 'A', 64 => '3', 65..126 => 'A'
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	1..63 => 'A', 64 => '4', 65..126 => 'A'
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	1..123 => 0, 124..126 => 298
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the	1..121 => 0, 122..126 => 690.0

maximum line length.

$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	1..63 => 'A'
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	1..62 => 'A', 63 => '1'
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	1..106 => ' '
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	2_097_152
$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	16
$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	SUN
$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	2#1.0#E-31
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	35
$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NO_SUCH_TYPE
$FLOAT_NAME The name of a predefined floating-point type other than	NO_SUCH_TYPE

FLOAT, SHORT_FLOAT, or LONG_FLOAT.	
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	200000.0
$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	31
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	ILLEGAL!@#$%^/ILLEGAL
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	ILLEGAL&()+-/ILLEGAL
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2_147_483_648
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-200000.0
$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	1

$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	126
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..2 => '2:', 3..123 => '0', 124..126 => '11:'
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..3 => '16:', 4..122 => '0', 123..126 => 'F.E:'
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	1 => '"', 2..125 => 'A', 126 => '"'
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	32
$NAME	NO_SUCH_TYPE

A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	
$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.	SUN
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFF_FFFF#
$NEW_MEM_SIZE An integer literal whose value is a permitted argument for pragma memory_size, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.	2_097_152
$NEW_STOR_UNIT An integer literal whose value is a permitted argument for pragma storage_unit, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	8
$NEW_SYS_NAME A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	IAPX386_PM
$TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has a single entry with one inout parameter.	32
$TICK A real literal whose value is SYSTEM.TICK.	1.0

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

A39005G
This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E
This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A
This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B
This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D
This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]
These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84M & N, & CD50110
These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C
These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B
This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B
This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]
These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A
This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK -- particular instances of change may be less (line 29).

CD7203B, & CD7204B
These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D
This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I
This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C
This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A
This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B
This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

E28005C
This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY

DDC-I, Inc

Compiler: DACS for Sun-3/SunOS, Version 4.4 (1.1)

ACVC Version: 1.10

OPTION	EFFECT

5 The Ada Compiler

The Ada Compiler is invoked by specifying a call of the program Ada to the shell. The invocation command is described in Section 5.1. on the Sun-3 workstation

If any diagnostic messages are produced during the compilation, they are output on the diagnostic file and on the standard output. The diagnostic file and the diagnostic messages are described in Sections 5.1.3 and 5.3.5.

The user may request additional listings to be output on a list file by specifying options in the compiler invocation. The list file and the listings are described in Sections 5.1.2 and 5.3.

The compiler uses a program library during the compilation. The compilation unit may refer to units from the program library, and an internal representation of the compilation unit will be stored in the program library as a result of a successful compilation. The program library is described in Chapter 3. Section 5.4 briefly describes how the Ada compiler uses the library.

5.1 The Invocation Command

The invocation command has the following syntax:

```
        ada  <source-file-name> {<source-file-name>}
```

Options

-L
-l

Causes the compiler to produce a formatted listing of the input source. The listing is written on the list file. Section 5.3.2 contains a description of the source listing. The default is no list file, in which case no source listing is produced, regardless of any LIST pragmas in the program or any diagnostic messages produced.

-x

Causes the compiler to produce a cross-reference listing. If this option is given and no severe or fatal errors are found during the compilation, the cross-reference listing will be written on the list file. The cross-reference listing is described in Section 5.3.4. The default excludes cross-reference.

-p

Progress-report.

-a <lib_spec>

Specifies the current sublibrary, and therefore the program library. If this option is omitted the sublibrary designated by the environment variable ADA_LIBRARY is used. If the variable does not exist the file ADA_LIBRARY is used. Section 5.4 describes how the Ada compiler uses the library.

-c <file_name>

Specifies the configuration file to be used by the compiler in the current compilation. If this option is omitted the configuration file (config) in the compiler directory is used.

-s
-S

Specifies that the source text is not to be saved in the program library. This saves some space in the sublibrary. The default is to save source text. In this way, the user is always certain what version of the source text was compiled. The source text may be displayed from the sublibrary with the PLU Type command.

-B

Build standard. Pseudo compilation of package standard. This option is intended for maintenance purposes only.

-n

No check. Suppress all run-time checks. By default, all run-time checks are generated.

-N <keyword> {,<keyword>}

Toggle check. Selective suppress of run-time checks. If a check is suppressed, the option will enable the check. If a check is enabled, the option will suppress the check. The following keywords are allowed:

- access
- index
- discriminant
- range
- length
- elaboration
- storage

Keywords are case-insensitive and can be abbreviated such that the abbreviation is unique.

-o Optimize. Optimize the program with respect to execution time, which, under normal circumstances, also is optimization with respect to size of the executable.

-O <keyword> [,<keyword>]
Toggle optimization. The correspondence between keywords and optimization is as follows:

Keyword	Optimization
Block	Optimize block and call frames.
Peep	Peephole optimization.
Cse	Common subexpressions elimination.
Reordering	Optimize aggregates and procedure calls.
Stackheight	Minimize stack height.
Fct2proc	Change functions to procedures.

The keywords are case-insensitive and can be abbreviated such that the abbrivation is unique.

-u <unit_number> Specifies that the compilation unit being compiled *is assigned* the unit number <unit_number> in the current sublibrary (see section 3.2.2 for explanation of unit numbers). This option will only work for:

- compilations containing a single compilation unit which is neither a subunit nor contains subunit stubs,

- unit numbers which are unused and follow the formula <unit_number> div 4096 = <sublibrary_level_number> where div is integer division and <sublibrary_level_number> is counted from the root to the current sublibrary by assigning the root the level number: 0 (zero). Thus legal unit numbers for the root sublibrary are 0...4095, for a child sublibrary of the root: 4096...8191 and so on.

Parameters

The <source-file-name> specifies the file containing the source texts to be compiled. A source file is expected to have the string ".ada" as the last four characters of its name. If the last part of the name does not contain ".", the string ".ada" is appended to the name. More than one file name must be specified.

5.1.1 The List File

The name of the list file is identical to the name of the source file except that the final characters ".ada" are replaced by ".lis". The list file will be placed in the current directory. The contents of the list file are described in Section 5.3.

5.1.2 The Diagnostic File

The name of the diagnostic file is identical to the name of the source file except that the final characters ".ada" are replaced by ".err". The diagnostic file will be placed in the invoker's current directory.

The diagnostic file is a file containing a list of diagnostic messages, each followed by a line showing the number of the line in the source that caused the message to be generated, and then by a blank line. The file is not separated into pages and there are no headings.

5.1.3 The Configuration File

Certain functional characteristics of the compiler may be modified by the user. These characteristics are passed to the compiler by means of a configuration file, which is a text file. The contents of the configuration file must be an Ada positional aggregate, written on one line, of the type CONFIGURATION_RECORD, which is described below. The configuration file is not accepted by the compiler in the following cases:

- The syntax does conform with the syntax for positional Ada aggregates.

- A value is outside the ranges specified below.

- A value is not specified as a literal.

- LINES_PER_PAGE is not greater than TOP_MARGIN + BOTTOM_MARGIN.

- The aggregate occupies more than one line.